Monadic combinator parsers work very similarly to the LL parsers we covered before, but the monadic interface manages the input stream for us. The resulting parsers are much easier to read and to write. Your goals are:

- Understand the types of the parser combinators.
- Explain the result of executing a parser.
- Explain the `<|>` combinator.
- Implement `many` and `many1`.

### Part 1 --- The Types

```hs
0 newtype Parser t = Parser (String -> [(t, String)])
1 run (Parser p) = p
2
3 oneOf xx =
4  Parser (\inp -> case inp of
5    (s:ss) | s `elem` xx -> [(s,ss)]
6    otherwise -> [])
7
8 sat pred =
9  Parser (\inp -> case inp of
10    (s:ss) | pred s -> [(s,ss)]
11    otherwise -> [])
12
13 p1 = run (oneOf "abc") "axy"
14 p2 = run (oneOf "abc") "xya"
```

The `newtype` is like `data`, but the resulting type has only one constructor, and it is optimized away by the compiler. We use it instead of simply saying `type Parser t = String -> [(t,String)]` because we can't declare a type as an instance, but we can declare a `newtype` as an instance.

**Problem 1** Review the code with your team and come to a consensus on what each part is doing. What will be the values of `p1` and `p2`?
**Problem 2)** Can you write the function `digit` that parses a digit? Use `sat` to do this. For more of a challenge, have it return an actual integer.

---

**Part 2 --- The Type Classes**

```haskell
instance Functor Parser where
    fmap f (Parser p) =
        Parser (inp -> [(f t, s) |
                         (t,s) <- p inp])

instance Applicative Parser where
    pure a = Parser (inp -> [(a,inp)])
    (Parser p1) <*> (Parser p2) =
        Parser (inp -> [(v1 v2, ss2) |
                         (v1,ss1) <- p1 inp,
                         (v2,ss2) <- p2 ss1])

instance Monad Parser where
    (Parser p) >>= f =
        Parser (inp -> concat [run (f v) inp' |
                                (v,inp') <- p inp])

data Exp = IntExp Integer
           | PlusExp Exp Exp
    deriving Show

p3 = run (IntExp <$> digit) "123"
p4 = run (PlusExp <$> getIntExp <*> getIntExp) "123"
p5 = do i1 <- getIntExp
       i2 <- getIntExp
       return (PlusExp i1 i2)
```

**Problem 3)** What is the value of `p3`? Trace through the evaluation and be sure everyone on your team understands how we got that result.

**Problem 4)** Write the function `getIntExp` that is like `digit` but encapsulates the digit in an `IntExp`. 
Problem 5) What is the value of $p_4$? Trace through the evaluation and be sure everyone on your team understands how we got that result.

Problem 6) What is the value of $p_5$? Trace through the evaluation and be sure everyone on your team understands how we got that result.

Part 3 --- Choice, Many, Many1

```
(Parser p1) <|> (Parser p2) =
  Parser (\inp -> take 1 $ p1 inp ++ p2 inp)

string [] = Parser (\inp -> [[],inp]])
string (s:ss) = do v <- char s
  vv <- string ss
  return $ v:vv

getPlusExp = do string "+"
  e1 <- getExp
  e2 <- getExp
  return (PlusExp e1 e2)

getExp = getIntExp
<|> getPlusExp
```

Problem 7) Examine the code for <|>. How does it work? Hint: consider the cases that $p_1$ succeeds, $p_1$ fails but $p_2$ succeeds, and both $p_1$ and $p_2$ fail.
Problem 8) Write the parsers many \( p \) and many1 \( p \) that take zero or more (for many) or one or more (for many1) repetitions of \( p \).

Problem 9) The way we suggested writing getIntExp only works for a single digit. Can you make it work for multi-digit integers now?

Part 4 --- Precedence

Problem 10) Modify PlusExp to be infix, and add TimesExp as well. Stratify the grammar so that TimesExp has higher precedence than PlusExp.
Manager | Keeps team on track
Recorder | Records decisions
Reporter | Reports to Class
Reflector | Assesses team performance

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?

Combinator Parsing Activity --- Team's Assessment (SII)

Manager or Reflector: Consider the objectives of this activity and your team's experience with it, and then answer the following questions after consulting with your team.

1. What was a strength of this activity? List one aspect that helped it achieve its purpose.

2. What is one thing we could do to improve this activity to make it more effective?

3. What insights did you have about the activity, either the content or at the meta level?